Demystifying the Resilience of Large Language Model Inference: An End-to-End Perspective

Yu Sun[1], Zachary Coalson[2], Shiyang Cheng[3],
Hang Liu[3], Sanghyun Hong[2], Zhao Zhang[3], Bo Fang[4*], Lishan Yang[1]
[1]*George Mason University, USA, {ysun23,lyang28}@gmu.edu*
[2]*Oregon State University, USA, {coalsonz,sanghyun.hong}@oregonstate.edu*
[3]*Rutgers University, USA, {sc2612,hl1097}@scarletmail.rutgers.edu, zhao.zhang@rutgers.edu*
[4]*Pacific Northwest National Lab, USA, bo.fang@pnnl.gov*

ABSTRACT

Deep neural networks are known to be resilient to random bitwise faults in their parameters. However, this resilience has primarily been established through studies of classification models. The extent to which this claim holds for large-language models remains under-explored. In this work, we conduct an extensive measurement study on the impact of random bitwise faults in commercial-scale language model inference. We first expose that these language models are not truly resilient to random bit-flips. While aggregate metrics such as accuracy may suggest resilience, an in-depth inspection of the generated outputs shows significant degradation in text quality. Our analysis also shows that tasks requiring more complex reasoning suffer more from performance and quality degradation. Moreover, we extend our resilience analysis to models with augmented reasoning capabilities, such as Chain-of-Thought or Mixture of Experts architectures.

CCS CONCEPTS

• **Computer systems organization** → **Reliability**; • **Computing methodologies** → **Natural language processing**.

KEYWORDS

Reliability, Large Language Models (LLMs), Resilience Assessment

ACM Reference Format:
Yu Sun[1], Zachary Coalson[2], Shiyang Cheng[3],, Hang Liu[3], Sanghyun Hong[2], Zhao Zhang[3], Bo Fang[4*], Lishan Yang[1]. 2025. Demystifying the Resilience of Large Language Model Inference: An End-to-End Perspective . In *The International Conference for High Performance Computing, Networking, Storage and Analysis (SC '25), November 16–21, 2025, St Louis, MO, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3712285.3759803

1 INTRODUCTION

Soft errors are the manifestation of transient faults at the system level, arising primarily due to radiation-induced bit flips or electromagnetic interference that alter the stored data or computational states without causing permanent hardware damage [10, 23, 55, 56]. With the increasing scale and complexity of high-performance computing (HPC) systems, the frequency and impact of soft errors have become more pronounced, posing critical challenges to system reliability and correctness [57, 76, 92].

Traditional error detection and correction techniques, which often rely on redundancy, checkpointing, or simple parity checks, can introduce substantial overhead in performance and power consumption, making them increasingly inefficient as systems scale up [4, 46, 53, 70]. Therefore, there is a critical need to better design fault tolerance solutions for HPC applications against soft errors, with application-specific error resilience characterization as a critical step towards this goal. A profound understanding of the application reaction to errors enables adaptive and cost-effective fault tolerance solutions.

Among applications consuming the major computation cycles of HPC systems, deep learning models, especially Large Language Models (LLMs) [72, 79, 86], present an unprecedented scale of the problem space. The error characteristics of such models depend on various factors beyond the numerical computation accuracy [2, 41, 71].

1. The multitasking nature of LLMs. Unlike traditional HPC workloads that focus strictly on numerical computations, LLM inference is inherently multitasking, simultaneously serving diverse applications such as natural language understanding, sentiment analysis, code generation, translation, and interactive conversational tasks [9, 54]. This multitasking nature results in complex and varied resilience, as each type of task may exhibit different sensitivities and responses to soft errors. Consequently, comprehensive resilience characterization must consider the varied operational contexts of LLMs.

2. Dataset-driven model behaviors. The LLM performance and resilience significantly depend on the datasets used for both pre-training and fine-tuning [85, 91]. Differences in dataset characteristics, such as complexity, semantic richness, and noise levels, lead to substantial variations in the model robustness to soft errors. Fine-tuning, which adapts pre-trained models to specific tasks, further affects resilience by altering model sensitivity based on domain-specific training data, adding complexity to error characterization.

3. Floating-point quantization. The extensive use of reduced-precision arithmetic (such as FP16 and INT8) to improve computational efficiency and reduce memory footprints might inherently alter the susceptibility of LLMs to soft errors. Lower-precision arithmetic reduces the margin for tolerable errors, while the valid range of floating-point values is also confined, making the effect on model outputs and predictions unclear [30, 50].

* Corresponding author.

SC '25, November 16–21, 2025, St Louis, MO, USA

ACM ISBN 979-8-4007-1466-5/2025/11.
https://doi.org/10.1145/3712285.3759803

4. New features of LLMs. Advanced LLM features, including Mixture of Experts (MoE) [20] and Chain-of-Thought (CoT) [82], dynamically adjust computational paths based on input data, context, or intermediate computational outcomes. This adaptive behavior increases the complexity of error resilience characterization because error propagation and the impact of the error become highly dependent on runtime conditions and decisions. Predicting how errors interact with such dynamically reconfigurations demands intricate, context-sensitive analysis.

These unique factors underscore the need for specialized and detailed error resilience characterization for LLMs, aiming to enable robust and effective fault tolerance solutions tailored specifically to their operational intricacies. To this end, we conduct statistical software-level fault injection to evaluate LLM reliability during inference phase on different types of tasks across several general-purpose models under both computational and memory fault models. We also explore the reliability impact of datatype, model quantization, and new features such as Mixture of Experts (MoE) and Chain-of-Thought (CoT). Our results demystify the prior studies and revisit the conclusion of "LLMs are quite resilient under transient faults" made by [2], highlighting the importance to the impact of soft errors. In short, we summarize our key observations:

- Memory faults are more problematic than computational faults. The underlying reason is the fault origin and propagation determined by unique computation pattern of LLMs.
- Generative tasks are more vulnerable than multiple-choice ones due to the error propagation in sequential generated tokens, especially for reasoning tasks.
- Fine-tuned LLMs for generative tasks are more reliable under memory faults, possibly because the training process enhances their ability to keep the sentence structure and fluency.
- MoE models are more vulnerable than dense models on multiple-choice tasks since the change of expert selection may leads to extra accuracy degradation. However, they are more reliable on generative tasks due to the lower possibility to use faulty expert in the following generation iterations.
- Beam search is more reliable than greedy search for generative tasks under computational faults because it can avoid corrupted tokens by altering to another potential token sequence.
- Using Chain-of-Thought increases the reliability on reasoning tasks because the model can recover from the corrupted tokens in the reasoning process.

2 BACKGROUND

In this section, we briefly introduce some background knowledge. We start with the general model architecture and then discuss several unique optimization techniques, including model quantization, Mixture of Experts, and Chain-of-Thought.

2.1 Architecture of Large Language Models

Large Language Models (LLMs) represent a significant breakthrough in artificial intelligence, transforming the field of natural language processing to many other domains. Most LLMs utilize transformer architectures [78], including encoders and/or decoders that leverage self-attention mechanisms. Encoders process input features, while decoders generate task-specific outputs based on these features. General-purpose LLMs typically employ decoder-only designs where causal masking restricts attention to preceding tokens, enhancing generation capabilities. Common LLMs consist of sequential transformer blocks, each containing an attention block and a Multi-Layer Perceptron (MLP). In the attention block, inputs are projected into Query (Q), Key (K), and Value (V) through linear transformations. Attention weights computed from Q-K interactions are applied to V, then processed through a final linear projection layer, out_proj. The transformer block typically consists of two-layer normalization operations positioned before and after the attention block. The MLP follows the second normalization layer and has three linear layers with an activation layer between them. This architecture aims to enhance the model representation capability after the attention mechanism. Figure 1 shows the architecture of LLama series models [77], which is widely used by most open-source general-purpose LLMs.

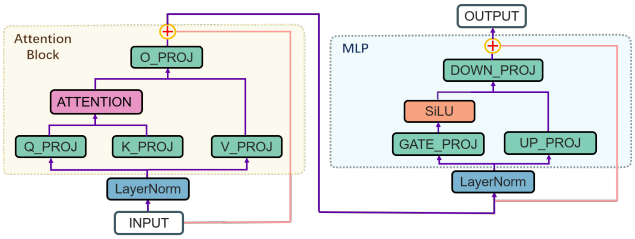


Figure 1: Llama3 model architecture (one transformer block).

As LLMs reach hundreds of billions of parameters, quantization reduces memory and computational requirements by converting high-precision weights (FP32) to lower bit-width formats (FP16, INT8, and INT4). Traditional quantization techniques like Post-Training Quantization (PTQ) [6, 47] and Quantization-Aware Training (QAT) [37] have been widely used for neural networks. Recent LLM-specific quantization methods include AWQ [43], ZeroQuant [89], and GPTQ [21]. These techniques enable significant memory reduction while preserving the performance[1].

2.2 Emerging LLM Optimizations

Mixture of Experts (MoE) is a powerful architecture introduced to improve model scaling efficiency by activating only a subset of parameters for each input [33]. It is applied to various domains including speech recognition [80], multi-modal learning [68], and machine translation [67]. In an MoE layer, inputs are first processed by a "router" or "gating network" that dynamically directs each token to a small set of specialized sub-networks called "experts". Each expert is a feed-forward network (i.e. MLP) specialized in processing different types of inputs. This sparse activation pattern allows MoE models to significantly increase parameter count with relatively low computational cost increases during inference. Recent LLM implementations have further refined MoE architectures, such as Mixtral [34], Deepseek [45], Gemini [73], and Llama4 [69]. By using only relevant parameters for each input, MoE models achieve higher performance compared to dense models with equivalent computational costs.

[1]In this paper we refer to performance as model performance, i.e., the ability of the model to generate satisfying outputs, not runtime performance, unless specifically mentioned.

Chain-of-Thought (CoT) is a prompting technique that enhances the reasoning capabilities of LLMs by encouraging them to generate intermediate reasoning steps before producing a final answer [82]. In CoT prompting, models are instructed by specific prompts such as "think step-by-step" or examples of detailed reasoning processes. This technique has proven effective for complex tasks requiring multi-step reasoning, such as mathematical problem-solving, logical reasoning, and code generation.

3 EXPERIMENTAL METHODOLOGY

In this section, we describe our experimental methodology. We start with the fault models and fault injection methods, then discuss the selection of LLM workloads and the experimental platform.

3.1 Fault Models

Soft errors are the manifestation of hardware transient faults originating from sources like cosmic radiation [22], shrinking transistors, and low voltage operations [10]. Here we consider both computational faults and memory faults. Computation-related faults affect computational hardware components such as ALUs (Arithmetic Logic Units), which aligns with the prior studies [5, 25, 40, 88]. We consider two typical computational fault types: 1) *1bit-comp:* single-bit flip and 2) *2bits-comp:* double-bit flip. As for memory faults, we assume that register files, caches, and memory are protected by Error Correction Code (ECC), which is the case in modern GPUs used for LLM tasks [27]. Hence, we only consider 3) *2bits-mem:* double-bit flip, which ECC cannot correct.

3.2 Fault Injection Methodology

Fault injection (FI) experiments are used to understand how hardware faults affect LLM inference. A fault injection experiment mimics the behavior of faults occurring in memory or computation paths, allowing us to observe the outcome of the fault [12, 39, 46].
Assumptions. We focus on the inference phase of LLMs since these models are typically trained once and then used repeatedly by different users [1, 11, 34, 72]. We assume there is only one error per inference because the probability of multiple faults occurring within the brief time frame of a single inference (which typically lasts only milliseconds or seconds) is extremely low. Our assumption is consistent with previous studies [2, 5, 19, 39, 40, 63, 66, 81].
Tool. Several fault injection frameworks for convolutional neural networks are publicly available, including TensorFI [13], PyTorchFI [51], and SNIFF [8]. For computational faults, our fault injection method utilizes the hook mechanism in PyTorch, which is similar to PyTorchFI. The hook function modifies the output tensor and the modified version is used in the following data path. For each fault injection trial, the location to inject a fault is identified by block ID, layer ID, neuron ID, and bit locations. We randomly choose a single token generation iteration for generative tasks. When injecting a 2-bit memory fault, we locate a weight position by the target block ID, layer ID, and weight ID, then randomly flip two bits in this weight before each inference instance. After each execution, we flip the same bits back to their fault-free values to enable a fresh execution for the next fault injection run. In detail, when deciding the target fault injection position, the block ID is randomly selected among all decoder blocks, and the layer ID is the type of the target linear layer. The weight/neuron ID is determined by the row and column number in the selected tensor.
Outcome category. For direct-answer tasks, such as multiple-choice and math, the outcome of a fault injection experiment can be categorized as 1) *Masked*, where the option or the final answer provided by the model is as same as the reference (ground truth), and 2) *Silent Data Corruption (SDC)*, where the final answer provided by the model is incorrect compared to the reference. For other tasks, such as translation, summarization, and question answering, we evaluate output quality with commonly used metrics [42, 58, 61].

One fault injection run reflects only the resilience of the specific injection location. Evaluating the resilience of the entire LLM requires exhaustive fault injection experiments on *every* fault location, which is infeasible given that the number of experiments can reach billions or even trillions. Instead, we conduct statistical fault injection, aligned with prior works [2, 87]. Each experiment involves injecting a single fault at a random (i.e., uniformly distributed) location. We only consider linear layers in the transformer blocks of the model, as they make up most of the computation (e.g., 94% in Llama2-7B model with sequence length set to 1024) and are therefore more likely to experience soft errors. We perform 1000 fault injection runs per input for multiple-choice and math tasks and 500–3000 for other generative tasks, totaling over 13 million fault injections for all evaluations, requiring approximately 4,800 GPU hours.

3.3 LLM Workload Selection

Table 1 lists all the considered downstream tasks, datasets, and models. We also present the metrics of output correctness evaluation.

3.3.1 Models. We focus on two sets of models: 1) general-purpose LLMs, which are well-suited for dealing with a wide range of tasks and are commonly used as base models for fine-tuning; 2) task-specific models, which are fine-tuned for better performance and efficiency in specialized tasks.

We select three representative general-purpose LLMs for characterization: 1) **Llama3.1** [26] is Meta AI's latest iteration of the Llama architecture, known for its improved reasoning capabilities and instruction-following. We use the 8B-Instruct version. 2) **Qwen2.5** [86] is Alibaba's advanced multilingual model with enhanced capabilities in non-English languages. We use the 1.5B/3B/7B-Instruct versions. 3) **Falcon3** [74] is Technology Innovation Institute's model trained on a diverse, multilingual corpus with strong performance on knowledge-intensive tasks. We employ the 7B-Instruct version.

We incorporate two additionalgyk task-optimized models: 1) **ALMA** [84] is specifically designed for multilingual translation tasks, fine-tuned on Llama2-7B model. 2) **Llama3.1-Summarizer** [52] is a fine-tuned variant of Llama3.1-8B model specifically for extractive summarization tasks. Moreover, to evaluate the reliability of MoE models, we employ Llama-3.2-8X3B-MOE-Instruct-18.4B model [16] and compare it with the corresponding dense model Llama-3.2-3B-Instruct.

3.3.2 Datasets. LLMs exhibit various capabilities including knowledge retrieval, reasoning, and content generation. We consider two sets of downstream tasks: 1) multiple-choice style tasks to test LLM

Table 1: Selected LLM workloads and metrics.

Tasks	Datasets	Metrics	Test models
General model understanding and reasoning	MMLU [29] AI2_ARC [14] TruthfulQA [44] WinoGrande [65] HellaSwag [90]	Accuracy	Llama3.1-8B [26] Qwen2.5-7B [86] Falcon3-7B [74]
Math	GSM8k [15]		Qwen2.5-7B Falcon3-7B
Translation	WMT16 [7]	BLEU [58] chrF++ [61]	Qwen2.5-7B Llama2-7B [77] ALMA-7B [84]
Summarization	XLSum [28]	Rouge-1 Rogue-L [42]	Llama3.1-8B Qwen2.5-7B Summarizer [52]
Question Answering	SQuAD v2 [62]	Exact Match F1 score	Llama3.1-8B Qwen2.5-7B Falcon3-7B

general reasoning and understanding and 2) generative tasks to evaluate LLM generative and reasoning capabilities. Due to the high computing resource consumption, we can only evaluate part of each dataset. To effectively and fairly select the subsets, we employ tinyBenchmarks [60], which provides standardized subsets (100 selected inputs) of these popular benchmarks.

Multiple-choice-style tasks and datasets are designed for evaluating the reasoning and knowledge capabilities of LLMs. During evaluation, the model scores each option and chooses the one with the highest score instead of generating content. We utilize 5 datasets: 1) **MMLU** (Massive Multitask Language Understanding) [29] assesses models across 57 subjects spanning mathematics, humanities, sciences, and more. 2) **ARC** (AI2 Reasoning Challenge) [14] evaluates grade-school level scientific reasoning abilities. 3) **TruthfulQA** [44] measures the model ability to avoid generating false or misleading information. 4) **WinoGrande** [65] tests commonsense reasoning through pronoun resolution challenges. 5) **HellaSwag** [90] evaluates commonsense reasoning abilities by asking models to choose the most plausible sentence completion.

In generative tasks, models are asked to produce content token by token. Their generative and reasoning capabilities are evaluated, including coherence, accuracy, and contextual appropriateness. We employ four datasets: 1) **GSM8k** [15] contains grade school math problems that test mathematical reasoning and step-by-step problem-solving. 2) **WMT16** [7] evaluates cross-lingual translation capabilities across different language pairs. Specifically, we use the de-en subset, which asks models to translate German to English. This is a common language pair that is included in most translation datasets. 3) **XLSum** [28] is a multilingual summarization dataset that tests the model ability to extract and condense key information. 4) **SQuAD v2** [62] is a question-answering dataset that evaluates the ability to extract answers from context.

3.3.3 Metrics. We evaluate model performance across various benchmarks using appropriate metrics.For multiple-choice tasks, we report accuracy percentages. Accuracy is used to evaluate the correctness of final answers for the math benchmark GSM8k. We use ROUGE-1 and ROUGE-L [42] metrics to assess content overlap and longest common subsequence matching for the XLSum summarization dataset. Machine translation quality on WMT16 is measured using BLEU [58] and chrF++ [61] scores. For the SQuAD v2 question-answering task, we report Exact Match and F1 Score percentages. As the baseline performance of different settings varies, we calculate normalized performance for each metric:

$$\text{Normalized Performance} = \frac{P_{\text{fault\_injected}}}{P_{\text{fault\_free}}},$$

where $P_{\text{fault\_injected}}$ is the metric value with fault injection and $P_{\text{fault\_free}}$ is the baseline performance without fault injection. We apply the Log-transformation method [35, 36] to obtain the error margins of normalized performance at 95% confidence level.

3.3.4 Generation Settings. For all experiments in evaluation section, we use the *generate()* function from huggingface [83] to perform inference. We disable sampling with a fixed random seed to ensure the same set of fault injection positions. We use greedy search by setting *num_beam=1*.

3.4 Experiment Platform

Due to limited computational resources, all experiments are conducted across two hardware configurations: an AMD EPYC 7742 64-Core CPU paired with an NVIDIA A100 GPU (Ampere architecture) running Rocky Linux 8.10, and an NVIDIA GH200 Grace Hopper Superchip that combines a NVIDIA Grace CPU with 72 Arm Neoverse V2 cores and an NVIDIA H100 GPU (Hopper architecture) running Rocky Linux 9.3. There are negligible differences ($< 0.03\%$ for raw model performance across these two hardware configurations).

4 CHARACTERIZATION RESULTS

We present an end-to-end characterization of LLM resilience, examining several key aspects across the entire workflow of LLM inference, as illustrated in Figure 2. We begin by analyzing the general resilience of LLMs under different fault models and inference tasks (subsection 4.1). Next, we explore how various model architectures and configurations influence resilience (subsection 4.2). Finally, we investigate the resilience implications of different operational settings during inference (subsection 4.3).

4.1 Overall Resilience Characteristics

Figure 3 presents the fault injection results for all tasks, datasets, and fault models considered in this study. The average model performance degradation is 2.28%, with the maximum observed degradation reaching 13.09%. Considerable performance improvement on the TruthfulQA dataset under computational faults is observed. This is different from the conventional neural networks where performance decreases under faults [12, 13, 48].

We observe up to 13.09% performance degradation for memory faults. As model sizes continue to grow, these larger architectures become increasingly vulnerable to memory faults during inference. The degree of resilience varies noticeably across different tasks, LLM architectures, and fault models. In the following sections, we delve deeper into these aspects, exploring and explaining the underlying reasons behind these resilience characteristics.

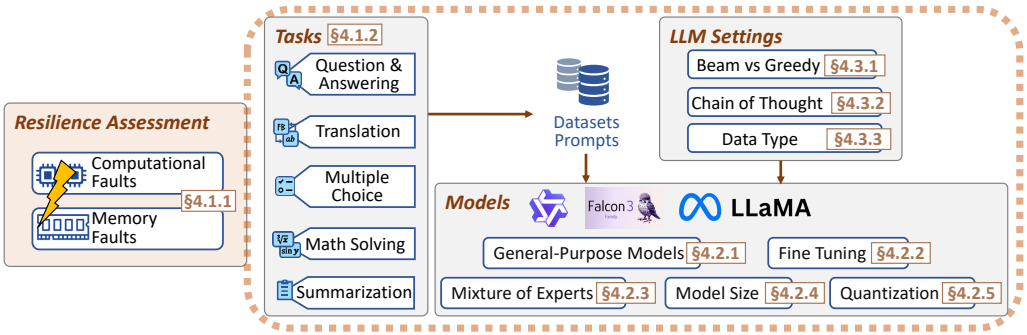
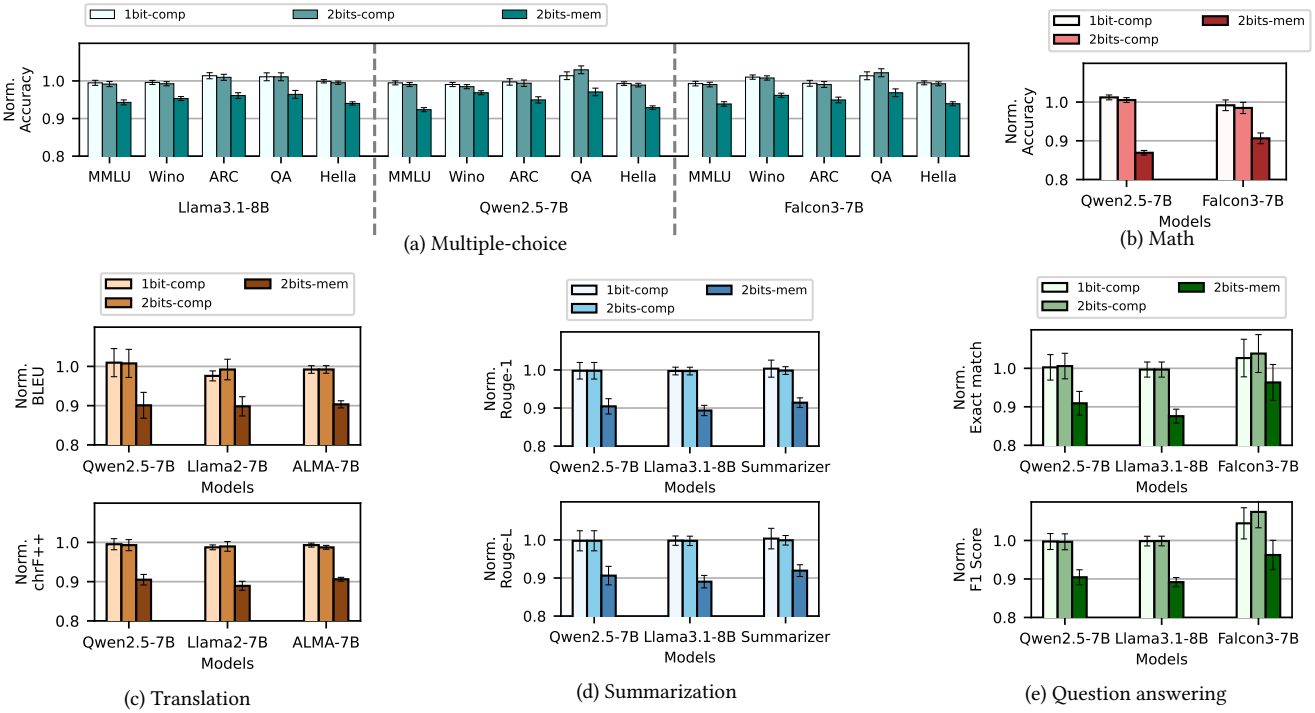


Figure 2: Workflow of LLM resilience assessment.



(a) Multiple-choice (b) Math (c) Translation (d) Summarization (e) Question answering

Figure 3: LLM performance change after fault injection. The numbers are normalized by the performance of the models under fault-free execution.

4.1.1 Resilience under Various Fault Models. The average performance under the considered fault models across all models and datasets is presented in Figure 4. Among these fault models, LLMs consistently demonstrate greater resilience to computational faults than to memory faults.

The observed resilience gap between these two fault types arises primarily due to differences in error origin and propagation behavior during computation. Examples of error propagation for a memory fault and a computational fault are illustrated in Figure 5 and Figure 6, respectively. Here, we inject a fault in the `up_proj` layer in Block10, flipping the most significant bit (MSB) at position (20, 20) of the weight/output tensor. We only show the first 50 × 50 elements in a tensor due to space constraints.

As shown in Figure 5, when a bit-flip occurs in memory, it corrupts a specific weight value utilized during GEMM (General Matrix-Matrix Multiplication) computations. This corrupted weight propagates errors across an entire column of the resulting output tensor.

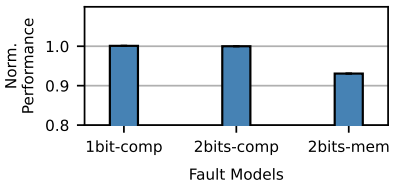


Figure 4: Average performance change of LLM workloads under different fault models. Memory faults cause higher performance degradation, thus, are more critical.

Since the output tensor becomes the input to subsequent layers, the corrupted column values further propagate, ultimately affecting the entire output tensor of the next layer.

Conversely, computational faults, modeled as bit-flip(s) occurring within neurons, exhibit a distinct propagation pattern. As illustrated in Figure 6, a computational fault initially propagates across a single

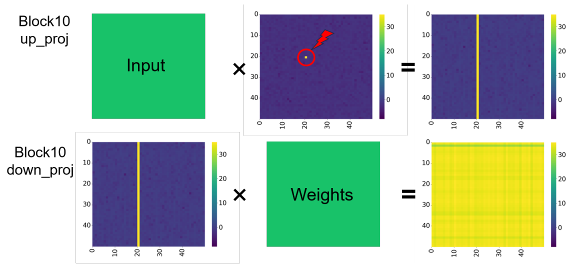


Figure 5: An example of the propagation trace of a memory fault. The fault propagates to the entire column of the output tensor of the fault-injected layer and then further propagates to the whole output tensor of the succeeding layer. Green tensors are fault-free ones; the heatmap shows the value distribution within a tensor (in log-scale), in which the yellow color indicates extremely large values caused by the fault.

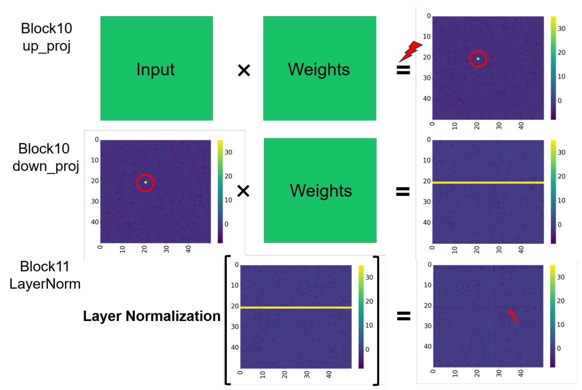


Figure 6: An example of a computational fault where it first propagates to the entire row of output tensor of the next layer, then is masked by the following normalization layer.

row of the output from the following layer and remains localized, unable to influence other rows until subsequent computations in the next transformer block. According to the transformer architecture (Figure 1), normalization layers typically precede each attention and MLP block, which can significantly mitigate and contain errors before they propagate widely. Due to the more localized and constrained propagation of computational faults compared to memory faults, the significant difference in resilience between these two fault types is well justified.

Analyzing the detailed outputs of fault injection experiments, the SDCs can be categorized into two types: 1) distorted outputs such as repeated or meaningless tokens and 2) subtly wrong outputs where the answer has incorrect information. Figure 7 shows some examples. The detailed breakdown of subtly wrong and distorted outputs is shown in Figure 8. Here we evaluate Qwen2.5-7B and Falcon3-7B models with GSM8k dataset. In most of the cases, the majority of SDCs are subtly wrong outputs, except Qwen2.5-7B under memory faults. Compared to computational faults, the occurrence of distorted outputs is more frequent under memory faults. This difference arises from the nature of error origin and propagation. For memory faults, 13.28% of outputs contain such distortions, while computational faults yield only 0.89–1.21% distorted outputs.

Furthermore, we dive into the understanding of the impact of faults affecting different bit positions. We group the experiments

Question: Question: Joe's bag of Halloween candy has 25 chocolate bars and 80 candied apples. Each chocolate bar weighs twice as much as each candied apple. If each chocolate bar weighs 40g, how much does Joe's bag of candy weigh, in grams?
Reference: 2600
Baseline: First find the weight of each candied apple: 40g / 2 = 20g. Then find the total weight of the chocolate bars: 25 bars * 40g / bar = 1000g. Then find the total weight of the candied apples: 80 apples * 20g / apple = 1600g. Then add the weight of the chocolate bars and candied apples to find the total weight: 1000g + 1600g = 2600g. Therefore, the answer is 2600.
Distorted output: !!
Subtly wrong output: First find the weight of each candied apple: 40g / 2 = 20g. Then find the total weight of the apple chocolate bars: 5 bars * 40g / bar = 200g. Then find the total weight of the apples: 80 apples * 20g / apple = 1600g. Then add the weight to find the total weight: 200g + 1600g = 1800g. The answer is 1800.

Figure 7: Examples of distorted and subtly wrong outputs.

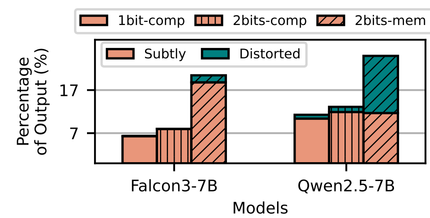


Figure 8: SDC breakdown of distorted and subtly wrong outputs with GSM8k dataset after fault injection. Subtly wrong outputs take the majority of the SDCs.

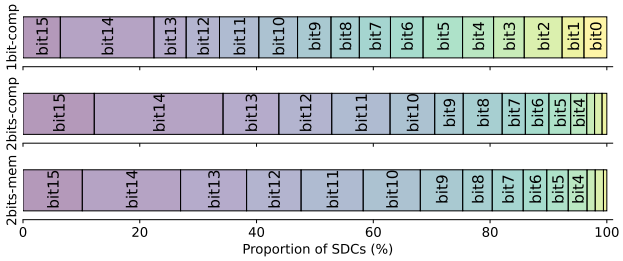
based on the position of the highest flipped bit, and then calculate the proportion of outputs contributed by each group. Figure 9 shows the distribution for subtly wrong outputs and Figure 10 shows the distribution for distorted outputs. In all cases, bit position 14 is the most vulnerable with the highest proportion, since it is the Most Significant Bit (MSB). Additionally, for distorted outputs, the highest two bits cause most of the distorted outputs, while the proportion is 0 for the mantissa bits.

> **Observation #1:** Memory faults are more problematic than computational faults. The underlying reason is the fault origin and propagation determined by the unique computation pattern of LLMs.

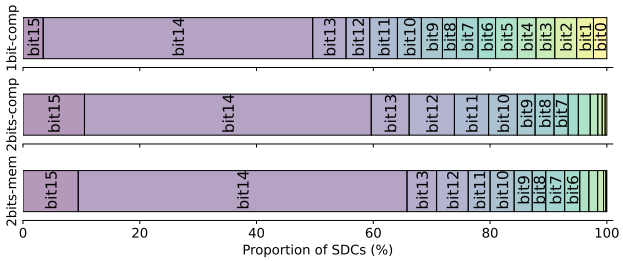
4.1.2 Resilience of Downstream Tasks. The average performance degradation after fault injection across various downstream tasks and datasets is presented in Figure 11. Among all evaluated tasks, TruthfulQA demonstrates the highest resilience, showing negligible performance degradation (with an average performance change of 0.04%). GSM8k, however, is the most vulnerable task, experiencing an average performance drop of approximately 3.85%.

A deeper examination of the results reveals that the observed differences in resilience are correlated with the nature of task types: multiple-choice versus generative tasks. Specifically, multiple-choice tasks exhibit an average accuracy drop of 1.65%, whereas generative tasks show a larger average decrease of 3.2%.

This gap is primarily due to the differences inherent in task evaluation methods. For multiple-choice tasks, faults may alter the confidence scores assigned to answer options, but such minor shifts can often be masked, allowing the correct option to remain selected. Generative tasks, in contrast, involve sequential token generation, causing faults to propagate through the output sequence, increasing

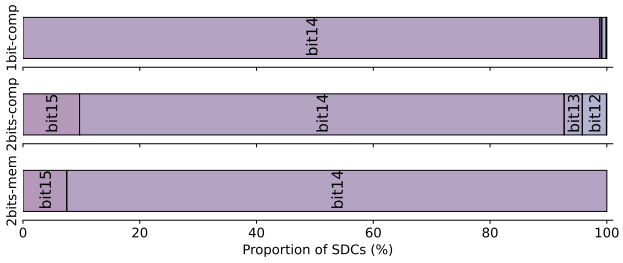


(a) Qwen2.5-7B

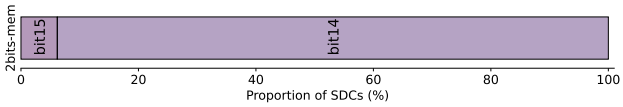


(b) Falcon3-7B

Figure 9: Proportion of subtly wrong outputs grouped by the position of the highest flipped bit. A wider range indicates that the corresponding bit position is more vulnerable. Bit position 14 is the most vulnerable bit.



(a) Qwen2.5-7B



(b) Falcon3-7B

Figure 10: Proportion of distorted outputs grouped by the position of the highest flipped bit. Bit position 14 is the most vulnerable one. Lower bits cannot cause distorted outputs. Note that here we do not show the corresponding results of computational faults for Falcon3-7B model, because the distorted outputs are very rare (< 0.1%).

the likelihood of generating incorrect tokens and ultimately causing Silent Data Corruptions (SDCs). Moreover, the impact of corrupted tokens varies across different generative tasks. For translation, summarization, and question-answering tasks, faults might reduce the output quality without completely corrupting it. Conversely, faults that occur in intermediate reasoning steps significantly increase the risk of generating incorrect final results for math-solving tasks, substantially degrading accuracy. An example illustrating how an intermediate calculation error caused by faults in the reasoning step propagates to the final output and leads to an SDC is shown in

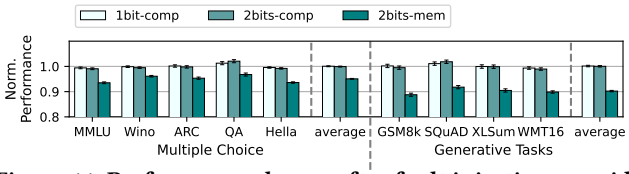


Figure 11: Performance change after fault injection, considering various downstream tasks with different datasets.

Question: Question: Raymond and Samantha are cousins. Raymond was born 6 years before Samantha. Raymond had a son at the age of 23. If Samantha is now 31, how many years ago was Raymond's son born?

Reference: 14

Baseline: Let's break it down step by step:
1. Samantha is currently 31 years old.
2. Raymond is 6 years older than Samantha, so Raymond is 31 + 6 = 37 years old.
3. Raymond had his son when he was 23 years old.To find out how many years ago Raymond's son was born, we need to subtract the age Raymond was when he had his son from Raymond's current age:
4. 37 (Raymond's current age) - 23 (age when Raymond had his son) = 14 years ago.
So, Raymond's son was born 14 years ago.

FI: Let's break it down step by step:
1. Samantha is currently 31 years old.
2. Raymond was born 6 years before Samantha, so Raymond is 31 - 6 = 25 years old.
3. Raymond had a son when he was 23 years old.To find out how many years ago Raymond's son was born, we need to subtract the age Raymond was when his son was born from Raymond's current age:
25 (Raymond's current age) - 23 (age when Raymond had his son) = 2
So, Raymond's son was born 2 years ago.
Therefore, the answer is 2 years ago.

Figure 12: An example of SDCs: A fault propagates and causes the LLM to generate different tokens in the reasoning process, leading to the wrong final answer. Critical tokens contributing to answer correctness are highlighted in red.

Figure 12. The "+" in step two changes to "-", propagating to the last reasoning step, and finally leads to an incorrect final answer.

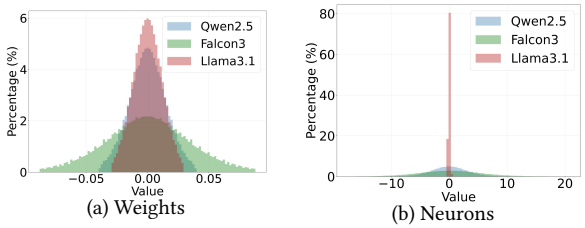
> **Observation #2:** Generative tasks are more vulnerable than multiple-choice ones due to the error propagation in sequential generated tokens, especially for reasoning tasks.

4.2 Model-Specific Resilience Study

We analyze the resilience characteristics across various model architectures and configurations. We cover general-purpose LLMs, fine-tuned models, Mixture-of-Experts, scale, and quantization.

4.2.1 Resilience Comparison of General-Purpose LLMs. Although the three general-purpose LLMs have similar architectures, their reliability is different and depends on certain experimental conditions. For example, Falcon3-7B experiences a 9.36% performance degradation on the GSM8k dataset under memory faults, compared to a larger degradation of 13.09% for Qwen2.5-7B. Conversely, Falcon3-7B exhibits a slightly higher degradation when subjected to computational faults. This indicates that the resilience of these foundation models varies according to the fault model. This observation is related to the percentage of distorted outputs: as shown in Figure 8, Falcon3-7B generates fewer distorted tokens compared to Qwen2.5-7B. Given that memory faults primarily drive the production of these distorted outputs, this difference helps explain the resilience gap between these two models.

For many tasks such as WinoGrande, TruthfulQA, and SQuAD v2, Falcon3-7B is more reliable than the others, as shown in Figure 3. The average performance degradations across all tasks and fault models are 2.08%, 1.92%, and 1% for Qwen2.5-7B, Llama3.1-8B, and Falcon3-7B, respectively. Although the general-purpose LLMs we



(a) Weights (b) Neurons

Figure 13: The value distributions of weights and neurons in the three studied general-purpose models. The varying distributions lead to the unique resilience behaviors.

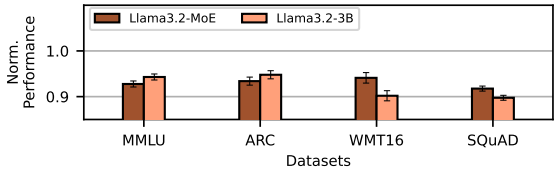


Figure 14: Performance degradation after fault injection comparing MoE models and normal models. MoE models are slightly more vulnerable than normal models on multiple-choice tasks, but more reliable on generative tasks.

evaluate share similar architectures, they are trained by different organizations using diverse datasets and proprietary techniques, making them distinct models. Precisely identifying the root causes of resilience differences remains challenging. However, the variations in their distributions of weights and neurons may provide insights. We illustrate these differences by examining the last linear layer (`down_proj`) of the last (28th) transformer block (Figure 13). The distribution for the three models is very different, which may lead to various distributions of values after bit flips. As Falcon3-7B has the widest distribution, the deviation after bit-flips could be smaller, leading to the high stability of the model.

> **Observation #3:** General-purpose LLMs exhibit different resilience to memory faults due to their varying distributions of neurons and weights.

4.2.2 Resilience Comparison of General-Purpose Models and Fine-Tuned Models. The right bars in Figure 3 (d) show the reliability of general-purpose models and fine-tuned models under memory faults. The fine-tuned Llama3.1-Summarizer exhibits greater resilience under memory faults compared with Llama3.1-8B. It is possible that fine-tuned generative LLMs are trained specially to maintain the structure and fluency of their outputs, which increases resilience to severe (memory) faults.

> **Observation #4:** Fine-tuned LLMs for generative tasks are more reliable under memory faults, possibly because the training process enhances their ability to maintain sentence structure and fluency.

4.2.3 Impact of Mixture-of-Experts. We evaluate LLama3.2-MoE model on MMLU, AI2_ARC, WMT16, and SQuAD v2 datasets since they include diverse content. The resilience impact of Mixture-of-Experts (MoE) architectures depends significantly on the type of task (Figure 14). For multiple-choice tasks (MMLU and ARC datasets), MoE models are slightly less resilient compared to dense models. However, for generative tasks like WMT16 and SQuAD v2, MoE models exhibit higher resilience.

Intuitively, given that only a subset of tokens are affected by a faulty expert, one might expect MoE models to be inherently more resilient. However, we observed scenarios where MoE models are more vulnerable. To understand these results, we analyze the underlying mechanism of expert selection. MoE models use router layers to dynamically assign experts (two out of eight experts in our experimental setup) to each token for each generation iteration. During the first iteration, approximately 25% of tokens might be directly affected by a memory fault. Due to error propagation across the entire column of the output tensor (as illustrated in Figure 5), faults affect all tokens (each row of the tensor). The faults further propagate to subsequent router layers, indirectly altering expert selections in later blocks. Since multiple-choice tasks typically involve only a single generation iteration, this initial widespread impact leads to slightly higher performance degradation in MoE models.

In contrast, generative tasks typically span multiple generation iterations. After the initial iteration, subsequent iterations usually rely on outputs from previous steps and cached key-value pairs, limiting the proportion of tokens directly influenced by a faulty expert. Thus, for generative tasks, the influence of faults diminishes significantly in subsequent iterations, resulting in overall higher resilience in MoE models compared to dense models.

> **Observation #5:** MoE models are more vulnerable than dense models on multiple-choice tasks since the change of expert selection may lead to extra accuracy degradation. However, they are more reliable on generative tasks due to the lower possibility of using faulty experts in the following generation iterations after the fault-injected layer.

MoE models introduce a specialized component, the gate layer (router), responsible for expert selection. Faults in gate layers affect the choice of experts, indirectly causing incorrect outputs. We use *model.children()* to locate the gate layer in each decoder block, which are between the attention blocks and experts. An example is shown in Figure 15, where we test the reliability of the MoE model on the translation task, with 2bits-mem faults injected in gate layers only. In 78.6% of the experiments, the expert's selections are changed, while 47.4% of those have an output with at least one changed token. The overall degradation of BLEU and chrF++ are 2.1% and 1.8%. Thus, gate layers present unique resilience considerations and must be explicitly protected. An attacker targeting gate layers could induce incorrect outputs without modifying the weights directly, highlighting its criticality from both reliability and security perspectives.

> **Observation #6:** Memory faults in gate layers can change the expert selections, leading to accuracy degradation without modifying the experts themselves.

4.2.4 The Impact of Model Scale. We evaluate the resilience of Qwen2.5 model with various sizes (1.5B, 3B, 7B, 14B, and 32B) on multiple datasets, see Figure 16. The evaluated models exhibit similar levels of reliability, and no clear relationship between model size and resilience emerges. Two factors potentially explain this observation. First, models within the same family share a common architectural foundation, resulting in similar error propagation patterns. Second, since these models originate from the same series, they likely employ similar training datasets and strategies. Thus,

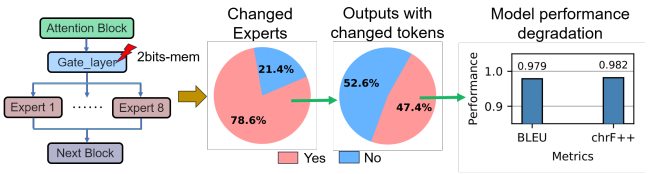


Figure 15: An example of computation faults leading to performance degradation when applying MoE, with Llama3.2-MoE model and WMT16 dataset. Faults in the gate layer may alter expert selection, leading to changes in generated tokens. This can cause SDCs and performance degradation.

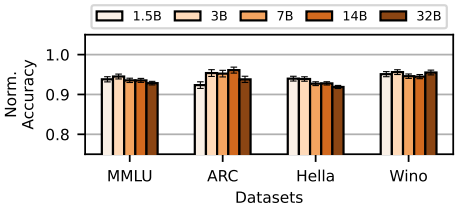


Figure 16: Performance change after fault injection in Qwen2.5 models with different model sizes. Model scale does not have a significant impact on LLM resilience.

differences in resilience among models of varying sizes within a single series are inherently limited. We conclude that model size alone is not a major factor influencing resilience for models within the same architecture family.

Observation #7: LLM scale does not affect model resilience.

4.2.5 Quantized. We evaluate two GPTQ [21] quantized variants (4-bit and 8-bit) of the Qwen2.5-7B model, comparing their resilience with the original BF16 version. For fault-free execution, model accuracy slightly decreases after quantization. Here we only evaluate the 2-bit memory fault model since the value and storage of quantized weights are the main differences compared with non-quantized models. The resilience results are shown in Figure 17. Model resilience increases after quantization. Both the 4-bit and 8-bit quantized models exhibit substantially greater resilience than the BF16 model, maintaining nearly 100% performance. This improvement is attributed to differences in numerical representation: for example, flipping the most significant exponent bit of a BF16 weight could drastically alter its magnitude (e.g., from 0.5 to approximately 1.7×10^{38}). In contrast, an equivalent bit-flip in a quantized 4-bit or 8-bit representation causes only a modest change in value, significantly reducing the likelihood of triggering an SDC.

Observation #8: Quantized models are more reliable, which is the opposite of intuition. Bit-flips within quantized weights cannot cause extreme deviation, lowering the chance of decreasing model performance.

4.3 Resilience Investigation of LLM Settings

4.3.1 Beam Search vs Greedy Search. We investigate the impact of different generation strategies (beam search versus greedy search) on model resilience using translation and summarization tasks, as shown in Figure 18. Since memory faults influence all token generation iterations and would dominate the resilience results, we

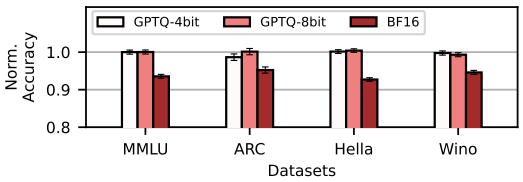
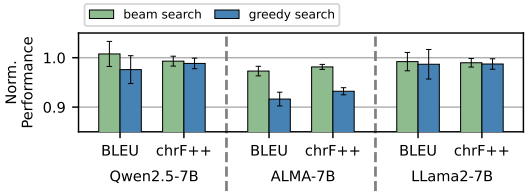
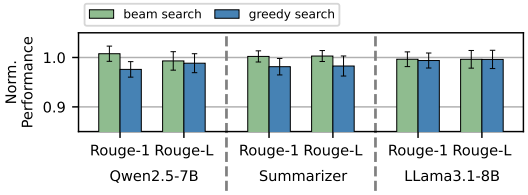


Figure 17: Resilience comparison of quantized models and non-quantized models in terms of performance change after fault injection. GPTQ-4bit and GPTQ-8bit are the two quantized versions of Qwen2.5-7B, compared to the non-quantized BF16 version. The quantized models are more reliable.



(a) WMT16 dataset



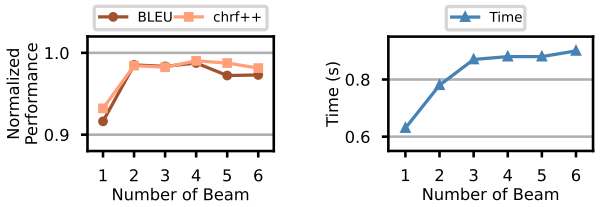
(b) XLSum dataset

Figure 18: Performance change after 2-bit computational fault injection, comparing the beam search and greedy search. Beam search is more resilient.

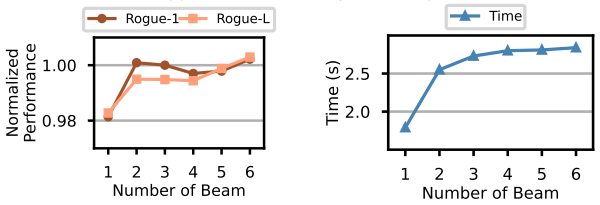
focus exclusively on the 2-bit computational fault model for this analysis. We set the beam size to 6, a commonly recommended value in prior work [24, 31]. Results indicate that beam search consistently demonstrates greater resilience than greedy search for fine-tuned models (ALMA-7B and Summarizer) on both tested datasets. Beam search can also increase the reliability of other models although the differences are not statistically significant. The real error margins for the employed metrics are narrower than those in Figure 18 since the value ranges are also narrower than the binomial distribution.

This resilience gap arises from fundamental differences between the two token selection algorithms. Greedy search selects the token with the highest confidence at each generation iteration, increasing vulnerability to errors; a single erroneous token selection can cascade, adversely affecting all subsequent token selections. Conversely, beam search explores multiple candidate sequences simultaneously, maintaining several potential generation paths based on cumulative probabilities. This approach provides robustness against isolated errors, as an erroneous token significantly lowers the cumulative probability of the affected path. Beam search can thus ultimately choose alternative paths unaffected by the faulty token, effectively mitigating computational faults.

Considering the trade-off between resilience and runtime overhead, we measure the resilience and runtime with different numbers of beams as shown in Figure 19. The setting with one beam refers



(a) WMT16 dataset (ALMA-7B)



(b) XLSum dataset (Summarizer)

Figure 19: The trade-off between resilience and runtime cost with different number of beams. Switching from greedy search (num_beam=1) to beam search, we notice an increase in normalized performance. When increasing the number of beams, the normalized performance shows no significant growth while the runtime cost continues to increase.

to the greedy search. When switching to beam search, the normalized performance increases, i.e., beam search is more reliable, with higher overhead. When the number of beams increases, the resilience remains stable, while the runtime continues to increase. Thus, the optimal trade-off point is setting the num_beam to 2.

In short, beam search is preferred for generative tasks to enhance resilience, especially when its adoption does not compromise generation accuracy or quality. When runtime overhead has to be minimized, greedy search should be used.

> **Observation #9:** Beam search is more reliable than greedy search for generative tasks with fine-tuned models under computational faults because it can avoid corrupted tokens by choosing another potential token sequence.

4.3.2 Reasoning: Chain-of-Thought (CoT). We assess the resilience of reasoning by evaluating the Qwen2.5-7B and Falcon3-7B models on GSM8k dataset with and without the reasoning process, i.e., generating the reasons. When evaluating the resilience of reasoning, the computational faults are only injected within the computation of generating reasoning tokens. For the injected memory faults, their impact lasts for the whole inference execution, including reasoning and the final answer generation. Both LLMs perform reasoning by default for math tasks. To disable reasoning, we add an instruction in the prompt to force the models to directly generate the final answer: "Solve the following math problem, but output only the final numerical answer".

Figure 20 shows the resilience evaluation results: using CoT is more reliable. Specifically, considering the computational faults, the normalized performance numbers are around 1.0 for both LLMs (the first two bars in each experiment set), indicating that computational faults in the reasoning part barely change the final answer. As for memory faults, although the normalized performance is around 0.9, using CoT still exhibits higher performance. This is because the model sometimes can recover from the wrong tokens during

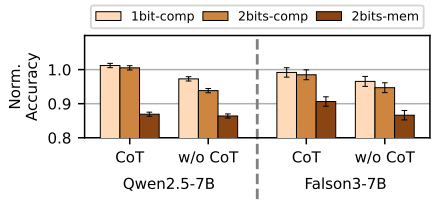


Figure 20: Performance change of Chain-of-Thought with fault injection. Applying CoT increases resilience.

the reasoning process, and still give out the correct final answer. In contrast, faults in the final answer generation have no further chance of being masked, thus are more likely to cause SDCs.

> **Observation #10:** Using CoT increases the reliability on reasoning tasks because the model can recover from the corrupted tokens in the reasoning process.

4.3.3 Data Type. We evaluate the impact of datatype on resilience for the Qwen2.5-7B model across several datasets (Figure 21). Our analysis indicates that FP16 datatype offers the highest resilience, while BF16 exhibits the greatest vulnerability.

Table 2: Format of floating-point data types.

Format	Total Bits	Exp Bits	Approximate Range
FP16	16	5	6×10^{-5} to 65504
FP32	32	8	10^{-38} to 3×10^{38}
BF16	16	8	10^{-38} to 3×10^{38}

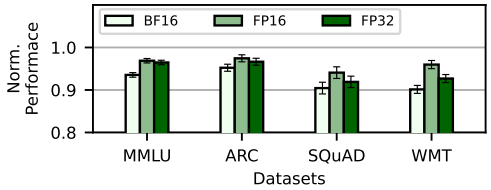


Figure 21: Performance degradation of Qwen2.5-7B after fault injection, considering various data types. FP16 is the most resilient datatype, while BF16 is the most vulnerable.

The observed vulnerability patterns across datatypes can be attributed to their respective bit allocations and representable numeric ranges, see Table 2. FP16 allocates 5 bits to the exponent, while BF16 allocates 8 exponent bits. Consequently, BF16 has a significantly larger representable numeric range compared to FP16, making bit-flips in its higher-order exponent bits more likely to result in extreme values and subsequent SDCs. When comparing FP32 and FP16, the representable range appears dominant in determining resilience: although FP16 has fewer bits and a higher probability that a random fault will affect exponent bits, the smaller exponent bit-width significantly limits the potential numeric impact of such faults, resulting in fewer extreme values and fewer SDCs.

> **Observation #11:** The datatype with a larger representable range and proportion of exponent bits is more unreliable.

5 RELATED WORK

Most existing studies focus on measuring reliability of neural networks [2, 17, 18, 32, 38, 39, 49, 59, 64, 75]. Li et al [39] conduct fault injection campaigns to analyze error propagation in DNNs, focusing on the input and outputs of GPU kernels. Ibrahim et al. [32] analyze

the resilience of Deep Residual Networks (ResNets) considering tasks such as object recognition and classification. DeepXplore [59] and DeepTest [75] evaluate the robustness of deep learning and autonomous driving systems, emphasizing the goal of finding corner cases and vulnerable contexts. Observations from DNNs cannot be directly applied to transformer-based LLMs due to architectural differences, such as the use of MoE and CoT in recent LLMs. Also, transformers rely on MLP (multi-layer perceptron), while CNNs and conventional DNNs incorporate convolution layers. It is essential to re-visit the resilience of LLMs. For transformers, Ma et al. combine fault injection and algorithmic checksum to assess the reliability of each component in transformer-based models [49]. Roquet et al. perform a beam test to measure fault effects on vision transformers [64]. The closest related work is by Agarwal et al. [2], who perform software-level fault injection to evaluate the reliability of LLMs across multiple tasks. Their study focuses on small-scale models with basic LLM architectures ranging from 38.9M to 222.9M parameters, using 10 inputs per dataset. To detect SDCs, they rely on strict word-to-word matching and cosine similarity, which may not fully capture the semantic correctness inherent to LLM tasks.

Different from all the above work, we perform an end-to-end complete study on LLM inference resilience, including various downstream tasks, model architectures, and settings, as well as different fault models. We consider emerging LLM performance optimization techniques such as MoE and CoT. Our characterization study reveals new insights that can guide the future design of AI-accelerators and LLMs. *To our best knowledge, this is the first end-to-end resilience study of LLM inference.*

6 LIMITATION

In this section, we discuss the limitations of our study. Firstly, the observations and conclusions in this paper may be restricted by the models we selected for evaluation. We focus on three popular open-source model architecture: llama, Qwen, and Falcon. We do not have access to closed-source models such as GPT4 [1], Gemini [73], and Claude [3]. We carefully select general-purpose and fine-tuned LLMs across multiple model families to maximize the generalizability of our findings. Additionally, due to limited computational resources, the experiments are performed on two hardware platforms equipped with NVIDIA GPUs. For the overlapping experiments we test on both platforms, there are negligible differences (less than 0.03% for raw model performance), indicating that our observations are not significantly affected by the hardware variation. Given that the fault injection study is performed at the software level without depending on the underlying architectural features, we expect similar observations on other accelerators such as AMD GPUs and AI ASICs. However, we cannot confirm our observations on those platforms due to lack of accesses. We believe our study provides valuable insights into LLM resilience to soft errors and establishes a solid foundation for future research in this critical area.

7 CONCLUSION

Modern HPC systems support exascale computations, enabling computationally demanding workloads like Large Language Model (LLM) inference. Yet, as scale increases, random bitwise faults occur in computing infrastructures. A common practice in handling these random faults is to *pass over* them, as studies have shown that neural networks are resilient to random bitwise corruptions. This work revisits this hypothesis and validates it through an extensive measurement study on the impact of random bitwise faults in LLMs. We examine three models trained on nine different datasets across five tasks and measure the impact using six metrics, each corresponding to a specific task. We also perform a manual analysis of the model generations. Moreover, we analyze the impact on models using techniques employed in typical deployment scenarios, e.g., quantization, Mixture of Experts, and Chain-of-Thought. Our results offer valuable insights for future research:

HPC system designers. We show that memory faults are more problematic than computational faults on LLMs. HPC system designers could therefore achieve greater resilience when running LLMs by protecting memory subsystems from random bitwise faults, rather than focusing more on reducing computational errors, such as silent data corruptions [57, 76].

LLM providers and algorithm developers. Our analysis suggests that the underlying cause of LLM susceptibility to random memory faults is the fault origin and propagation, which are determined by the unique computation patterns of LLMs. Hence, future work could focus on developing inference algorithms for LLMs that reduce fault propagation (i.e., fault isolation) and enhance model resilience. Generative tasks are more vulnerable than multiple-choice tasks due to error propagation in sequentially generated tokens, especially for reasoning tasks. We thus encourage future work on designing performance metrics that characterize the quality degradation of generated outputs across various aspects.

Practitioners. Our study finds that specialized LLMs, through fine-tuning, are more reliable under memory faults compared to general-purpose LLMs. Therefore, when employing LLMs for specific tasks in application development, it would be desirable to fine-tune general-purpose LLMs and enhance their resilience under random bitwise faults. Generally, MoE and CoT models are more reliable on generative and reasoning tasks. But when using MoE for multiple-choice tasks, extra care must be taken, as it makes the LLMs more vulnerable than their standalone counterparts.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation (NSF) grants (#2402940, #2410856, and #2417750) and the Commonwealth Cyber Initiative (CCI) grant (#HC-3Q24-047). The platforms used for evaluation in this work are supported by the U.S. DOE Office of Science, Office of Advanced Scientific Computing Research, under award 66150: "CENATE - Center for Advanced Architecture Evaluation". The Pacific Northwest National Laboratory is operated by Battelle for the U.S. Department of Energy under Contract DE-AC05-76RL01830. This project is supported by resources provided by the Office of Research Computing at George Mason University (URL: https://orc.gmu.edu) and funded in part by grants from the National Science Foundation (Award Number 2018631). The authors acknowledge the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing computational resources that have contributed to the research results reported within this paper. URL: http://www.tacc.utexas.edu.

REFERENCES

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[2] Udit Kumar Agarwal, Abraham Chan, and Karthik Pattabiraman. 2023. Resilience Assessment of Large Language Models under Transient Hardware Faults. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 659–670.

[3] Anthropic. 2024. The Claude 3 Model Family: Opus, Sonnet, Haiku. https://api.semanticscholar.org/CorpusID:270640496

[4] Ali Asgari Khoshouyeh, Florian Geissler, Syed Qutub, Michael Paulitsch, Prashant Nair, and Karthik Pattabiraman. 2023. Structural coding: A low-cost scheme to protect cnns from large-granularity memory faults. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–17.

[5] Rizwan A Ashraf, Roberto Gioiosa, Gokcen Kestor, Ronald F DeMara, Chen-Yong Cher, and Pradip Bose. 2015. Understanding the propagation of transient errors in HPC applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12.

[6] Ron Banner, Yury Nahshan, and Daniel Soudry. 2019. Post training 4-bit quantization of convolutional networks for rapid-deployment. *Advances in Neural Information Processing Systems* 32 (2019).

[7] Ondrej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, et al. 2016. Findings of the 2016 conference on machine translation (wmt16). In *First conference on machine translation*. Association for Computational Linguistics, 131–198.

[8] Jakub Breier, Dirmanto Jap, Xiaolu Hou, Shivam Bhasin, and Yang Liu. 2021. SNIFF: reverse engineering of neural networks with fault attacks. *IEEE Transactions on Reliability* 71, 4 (2021), 1527–1539.

[9] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[10] Nandhini Chandramoorthy, Karthik Swaminathan, Martin Cochet, Arun Paidimarri, Schuyler Eldridge, Rajiv V Joshi, Matthew M Ziegler, Alper Buyuktosunoglu, and Pradip Bose. 2019. Resilient low voltage accelerators for high energy efficiency. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 147–158.

[11] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[12] Zitao Chen, Guanpeng Li, and Karthik Pattabiraman. 2021. A low-cost fault corrector for deep neural networks through range restriction. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 1–13.

[13] Zitao Chen, Niranjhana Narayanan, Bo Fang, Guanpeng Li, Karthik Pattabiraman, and Nathan DeBardeleben. 2020. Tensorfi: A flexible fault injection framework for tensorflow applications. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 426–435.

[14] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457* (2018).

[15] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training Verifiers to Solve Math Word Problems. *arXiv preprint arXiv:2110.14168* (2021).

[16] DavidAU. 2025. *Llama-3.2-8X3B-MOE-Dark-Champion-Instruct-uncensored-abliterated-18.4B*. Hugging Face. https://huggingface.co/DavidAU/Llama-3.2-8X3B-MOE-Dark-Champion-Instruct-uncensored-abliterated-18.4B

[17] Fernando Fernandes dos Santos, Caio Lunardi, Daniel Oliveira, Fabiano Libano, and Paolo Rech. 2019. Reliability evaluation of mixed-precision architectures. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 238–249.

[18] Bo Fang, Xinyi Li, Harvey Dam, Cheng Tan, Siva Kumar Sastry Hari, Timothy Tsai, Ignacio Laguna, Dingwen Tao, Ganesh Gopalakrishnan, Prashant Nair, Kevin Barker, and Ang Li. 2024. Understanding Mixed Precision GEMM with MPGemmFI: Insights into Fault Resilience . In *2024 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE Computer Society, Los Alamitos, CA, USA, 166–178. https://doi.org/10.1109/CLUSTER59578.2024.00022

[19] Bo Fang, Karthik Pattabiraman, Matei Ripeanu, and Sudhanva Gurumurthi. 2014. GPU-Qin: A methodology for evaluating the error resilience of GPGPU applications. In *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*. IEEE, 221–230.

[20] William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research* 23, 120 (2022), 1–39.

[21] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323* (2022).

[22] Vinicius Fratin, Daniel A. G. de Oliveira, Caio B. Lunardi, Fernando Santos, Gennaro Rodrigues, and Paolo Rech. 2018. Code-Dependent and Architecture-Dependent Reliability Behaviors. In *DSN*. 13–26.

[23] Vinicius Fratin, Daniel Oliveira, Caio Lunardi, Fernando Santos, Gennaro Rodrigues, and Paolo Rech. 2018. Code-dependent and architecture-dependent reliability behaviors. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 13–26.

[24] Markus Freitag and Yaser Al-Onaizan. 2017. Beam search strategies for neural machine translation. *arXiv preprint arXiv:1702.01806* (2017).

[25] Giorgis Georgakoudis, Ignacio Laguna, Dimitrios S Nikolopoulos, and Martin Schulz. 2017. Refine: Realistic fault injection via compiler-based instrumentation for accuracy, portability and speed. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–14.

[26] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024).

[27] Hui Guan, Lin Ning, Zhen Lin, Xipeng Shen, Huiyang Zhou, and Seung-Hwan Lim. 2019. In-place zero-space memory protection for cnn. *Advances in Neural Information Processing Systems* 32 (2019).

[28] Tahmid Hasan, Abhik Bhattacharjee, Md Saiful Islam, Kazi Samin, Yuan-Fang Li, Yong-Bin Kang, M Sohel Rahman, and Rifat Shahriyar. 2021. XL-sum: Large-scale multilingual abstractive summarization for 44 languages. *arXiv preprint arXiv:2106.13822* (2021).

[29] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300* (2020).

[30] Sanghyun Hong, Pietro Frigo, Yiğitcan Kaya, Cristiano Giuffrida, and Tudor Dumitraș. 2019. Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks. In *28th USENIX Security Symposium (USENIX Security 19)*. 497–514.

[31] Hugging Face. [n. d.]. Generation strategies. Hugging Face Documentation. https://huggingface.co/docs/transformers/generation_strategies

[32] Younis Ibrahim, Haibin Wang, Man Bai, Zhi Liu, Jianan Wang, Zhiming Yang, and Zhengming Chen. 2020. Soft Error Resilience of Deep Residual Networks for Object Recognition. *IEEE Access* 8 (2020), 19490–19503.

[33] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. 1991. Adaptive mixtures of local experts. *Neural computation* 3, 1 (1991), 79–87.

[34] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088* (2024).

[35] Harold A Kahn and Christopher T Sempos. 1989. *Statistical methods in epidemiology*. Number 12. Monographs in Epidemiology and.

[36] DJSM Katz, J Baptista, SP Azen, and MC Pike. 1978. Obtaining confidence intervals for the risk ratio in cohort studies. *Biometrics* (1978), 469–474.

[37] Raghuraman Krishnamoorthi. 2018. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342* (2018).

[38] Jonathan Lew, Deval A Shah, Suchita Pati, Shaylin Cattell, Mengchi Zhang, Amruth Sandhupatla, Christopher Ng, Negar Goli, Matthew D Sinclair, Timothy G Rogers, et al. 2019. Analyzing machine learning workloads using a detailed GPU simulator. In *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 151–152.

[39] Guanpeng Li, Siva Kumar Sastry Hari, Michael Sullivan, Timothy Tsai, Karthik Pattabiraman, Joel Emer, and Stephen W Keckler. 2017. Understanding error propagation in deep learning neural network (DNN) accelerators and applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12.

[40] Guanpeng Li and Karthik Pattabiraman. 2018. Modeling input-dependent error propagation in programs. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 279–290.

[41] Yuhang Liang, Xinyi Li, Jie Ren, Ang Li, Bo Fang, and Jieyang Chen. 2025. ATTNChecker: Highly-Optimized Fault Tolerant Attention for Large Language Model Training. In *Proceedings of the 30th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*. 252–266.

[42] Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*. 74–81.

[43] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of Machine Learning and Systems* 6 (2024), 87–100.

[44] Stephanie Lin, Jacob Hilton, and Owain Evans. 2021. Truthfulqa: Measuring how models mimic human falsehoods. *arXiv preprint arXiv:2109.07958* (2021).

[45] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437* (2024).

[46] Haoxuan Liu, Vasu Singh, Michał Filipiuk, and Siva Kumar Sastry Hari. 2024. ALBERTA: ALgorithm-Based Error Resilience in Transformer Architectures. *IEEE Open Journal of the Computer Society* (2024).

[47] Zhenhua Liu, Yunhe Wang, Kai Han, Wei Zhang, Siwei Ma, and Wen Gao. 2021. Post-training quantization for vision transformer. *Advances in Neural Information Processing Systems* 34 (2021), 28092–28103.

[48] Dongning Ma, Fred Lin, Alban Desmaison, Joel Coburn, Daniel Moore, Sriram Sankar, and Xun Jiao. 2024. Dr. DNA: Combating Silent Data Corruptions in Deep Learning using Distribution of Neuron Activations. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. 239–252.

[49] Kwondo Ma, Chandramouli Amarnath, and Abhijit Chatterjee. 2023. Error Resilient Transformers: A Novel Soft Error Vulnerability Guided Approach to Error Checking and Suppression. In *2023 IEEE European Test Symposium (ETS)*. IEEE, 1–6.

[50] A. Mahmoud, N. Aggarwal, A. Nobbe, J. R. S. Vicarte, S. V. Adve, C. W. Fletcher, I. Frosio, and S. K. S. Hari. 2020. PyTorchFI: A Runtime Perturbation Tool for DNNs. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. 25–31.

[51] Abdulrahman Mahmoud, Neeraj Aggarwal, Alex Nobbe, Jose Rodrigo Sanchez Vicarte, Sarita V Adve, Christopher W Fletcher, Iuri Frosio, and Siva Kumar Sastry Hari. 2020. Pytorchfi: A runtime perturbation tool for dnns. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 25–31.

[52] Meta and RAAEC. 2024. Meta-Llama-3.1-8B-Instruct-Summarizer. https://huggingface.co/raaec/Meta-Llama-3.1-8B-Instruct-Summarizer. Accessed: 2025-04-14.

[53] Andrew Milluzzi and Alan George. 2017. Exploration of TMR fault masking with persistent threads on Tegra GPU SoCs. In *2017 IEEE Aerospace Conference*. IEEE, 1–7.

[54] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Nick Barnes, and Ajmal Mian. 2023. A comprehensive overview of large language models. *arXiv preprint arXiv:2307.06435* (2023).

[55] Bin Nie, Devesh Tiwari, Saurabh Gupta, Evgenia Smirni, and James H Rogers. 2016. A large-scale study of soft-errors on GPUs in the field. In *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*. IEEE, 519–530.

[56] Bin Nie, Ji Xue, Saurabh Gupta, Christian Engelmann, Evgenia Smirni, and Devesh Tiwari. 2017. Characterizing Temperature, Power, and Soft-Error Behaviors in Data Center Systems: Insights, Challenges, and Opportunities. In *MASCOTS 2017*. 22–31.

[57] Vladyslav Oles, Anna Schmedding, George Ostrouchov, Woong Shin, Evgenia Smirni, and Christian Engelmann. 2024. Understanding GPU Memory Corruption at Extreme Scale: The Summit Case Study. In *Proceedings of the 38th ACM International Conference on Supercomputing*. 188–200.

[58] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 311–318.

[59] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2019. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. *Commun. ACM* 62, 11 (oct 2019), 137–145.

[60] Felipe Maia Polo, Lucas Weber, Leshem Choshen, Yuekai Sun, Gongjun Xu, and Mikhail Yurochkin. 2024. tinyBenchmarks: evaluating LLMs with fewer examples. *arXiv preprint arXiv:2402.14992* (2024).

[61] Maja Popović. 2017. chrF++: words helping character n-grams. In *Proceedings of the second conference on machine translation*. 612–618.

[62] Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know What You Don't Know: Unanswerable Questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Iryna Gurevych and Yusuke Miyao (Eds.). Association for Computational Linguistics, Melbourne, Australia, 784–789. https://doi.org/10.18653/v1/P18-2124 arXiv:1806.03822 [cs.CL]

[63] Brandon Reagen, Udit Gupta, Lillian Pentecost, Paul Whatmough, Sae Kyu Lee, Niamh Mulholland, David Brooks, and Gu-Yeon Wei. 2018. Ares: A framework for quantifying the resilience of deep neural networks. In *Proceedings of the 55th Annual Design Automation Conference*. 1–6.

[64] Lucas Roquet, Fernando Fernandes dos Santos, Paolo Rech, Marcello Traiola, Olivier Sentieys, and Angeliki Kritikakou. 2024. Cross-Layer Reliability Evaluation and Efficient Hardening of Large Vision Transformers Models. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*. 1–6.

[65] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Commun. ACM* 64, 9 (2021), 99–106.

[66] Christoph Schorn, Andre Guntoro, and Gerd Ascheid. 2018. Efficient on-line error detection and mitigation for deep neural network accelerators. In *Computer Safety, Reliability, and Security: 37th International Conference, SAFECOMP 2018, Västerås, Sweden, September 19-21, 2018, Proceedings 37*. Springer, 205–219.

[67] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538* (2017).

[68] Yuge Shi, Brooks Paige, Philip Torr, et al. 2019. Variational mixture-of-experts autoencoders for multi-modal deep generative models. *Advances in neural information processing systems* 32 (2019).

[69] Ajit Singh. 2025. Meta Llama 4: The Future of Multimodal AI. *Available at SSRN 5208228* (2025).

[70] Charles W Slayman. 2005. Cache and memory error detection, correction, and reduction techniques for terrestrial servers and workstations. *IEEE Transactions on Device and Materials Reliability* 5, 3 (2005), 397–404.

[71] Yu Sun, Zhu Zhu, Cherish Mulpuru, Roberto Gioiosa, Zhao Zhang, Bo Fang, and Lishan Yang. 2025. FT2: First-Token-Inspired Online Fault Tolerance on Critical Layers for Generative Large Language Models. (2025).

[72] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805* (2023).

[73] Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530* (2024).

[74] TII Team. 2024. The Falcon 3 family of Open Models.

[75] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. DeepTest: Automated Testing of Deep-Neural-Network-Driven Autonomous Cars. In *Proceedings of ICSE'18*. 303–314.

[76] Devesh Tiwari, Saurabh Gupta, George Gallarno, Jim Rogers, and Don Maxwell. 2015. Reliability lessons learned from GPU experience with the Titan supercomputer at Oak Ridge leadership computing facility. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2015, Austin, TX, USA, November 15-20, 2015*. 38:1–38:12. https://doi.org/10.1145/2807591.2807666

[77] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971 [cs.CL]

[78] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[79] Ben Wang and Aran Komatsuzaki. 2021. GPT-J-6B: A 6 billion parameter autoregressive language model.

[80] Steven Richard Waterhouse. 1998. *Classification and regression using mixtures of experts*. Ph. D. Dissertation. Citeseer.

[81] Jiesheng Wei, Anna Thomas, Guanpeng Li, and Karthik Pattabiraman. 2014. Quantifying the accuracy of high-level fault injection techniques for hardware faults. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 375–382.

[82] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[83] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771* (2019).

[84] Haoran Xu, Young Jin Kim, Amr Sharaf, and Hany Hassan Awadalla. 2023. A paradigm shift in machine translation: Boosting translation performance of large language models. *arXiv preprint arXiv:2309.11674* (2023).

[85] Runxin Xu, Fuli Luo, Zhiyuan Zhang, Chuanqi Tan, Baobao Chang, Songfang Huang, and Fei Huang. 2021. Raise a child in large language model: Towards effective and generalizable fine-tuning. *arXiv preprint arXiv:2109.05687* (2021).

[86] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. 2024. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115* (2024).

[87] Lishan Yang, Bin Nie, Adwait Jog, and Evgenia Smirni. 2021. Practical Resilience Analysis of GPGPU Applications in the Presence of Single- and Multi-Bit Faults. *IEEE Trans. Comput.* 70, 1 (2021), 30–44.

[88] Lishan Yang, Bin Nie, Adwait Jog, and Evgenia Smirni. 2021. SUGAR: Speeding Up GPGPU Application Resilience Estimation with Input Sizing. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 5, 1 (2021), 1–29.

[89] Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. 2022. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in Neural Information Processing*

Systems 35 (2022), 27168–27183.

[90] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830* (2019).

[91] Shengyu Zhang, Linfeng Dong, Xiaoya Li, Sen Zhang, Xiaofei Sun, Shuhe Wang, Jiwei Li, Runyi Hu, Tianwei Zhang, Fei Wu, et al. 2023. Instruction tuning for large language models: A survey. *arXiv preprint arXiv:2308.10792* (2023).

[92] Zhu Zhu, Yu Sun, Dhatri Parakal, Bo Fang, Steven Farrell, Gregory H Bauer, Brett Bode, Ian T Foster, Michael E Papka, William Gropp, et al. 2025. Understanding the Landscape of Ampere GPU Memory Errors. *arXiv preprint arXiv:2508.03513* (2025).